Introduction to Computer Organization – Fall 2023

Homework 2

Due: @11:55pm, Monday October 9th on Gradescope

Name: Yuzhen Chen Uniqname: yuzhench

1. Submit a pdf of your typed or handwritten homework on Gradescope.

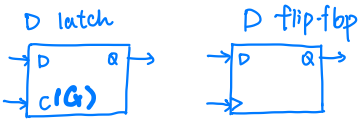
2. Your answers should be neat, clearly marked, and concise. Typed work is recommended, but not required unless otherwise stated. Show all your work where requested, and state any special or non-obvious assumptions you make.

3. You may discuss your solution methods with other students, but the solutions you submit must be your own.

4. **Late Homework Policy:** Submissions turned in by 1:00 am the next day will be accepted but with a 5% penalty. Assignments turned in between 1:00 am and 11:55 pm will get a 30% penalty, and any submissions made after this time will not be accepted.

5. When submitting your answers to Gradescope you need to indicate what page(s) each problem is on to receive credit. The grader may choose not to grade the homework if answer locations are not indicated.

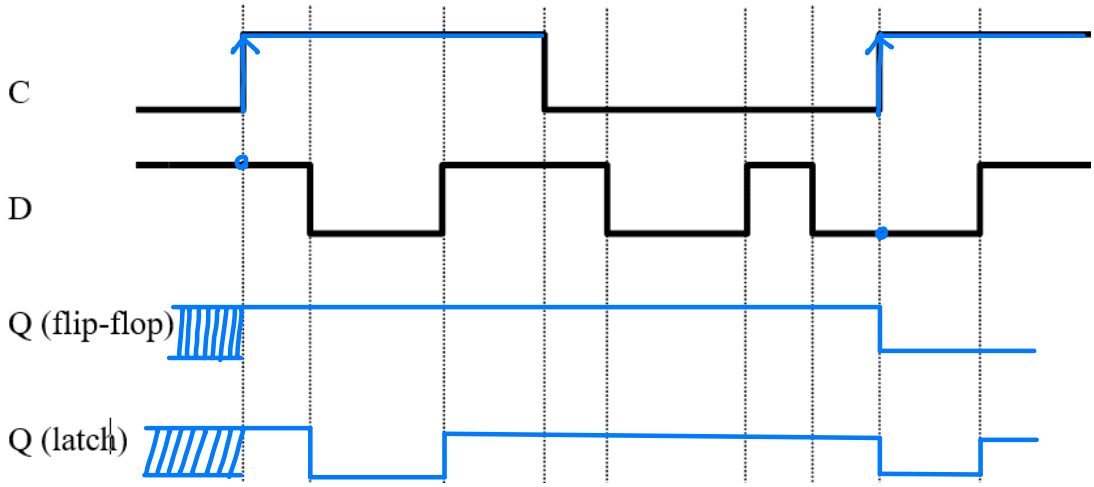
6. **The last two questions are group questions**.
    - **You will turn those questions in separately** and may do it in a group of up to two students (yes, you can do it by yourself if you wish).
    - It is an honor code violation if a student is listed as contributing who did not actually participate in working on that problem.
    - Further, we suggest that you not split this up but rather work on the problem as a group.
7. After each question (or in some cases question part), we've indicated which lecture number we expect to cover the relevant material. So "**(L7)**" indicates that we expect to cover the material in lecture 7.



Problem 1: Latches and flip-flops [8 points]

C=0 Q=Q' C=1 Q=D

Complete the timing diagrams for a D latch and D flip-flop given the provided inputs. The flip-flop is a positive-edge triggered. For the latch we are using "C" for the "G" input (which is fairly common). Assume both the latch and the flip-flop have an initial value of zero. **(L9)**



Problem 2: State machines [8 points]

Answer the following questions about state machines. **(L9)**

a) Say you have a state machine with 22 states, what is the minimum number of bits you would need to store the current state of that state machine? **[2]**

$\lceil \log_2^{22} \rceil = 5$ So we need 5 bits

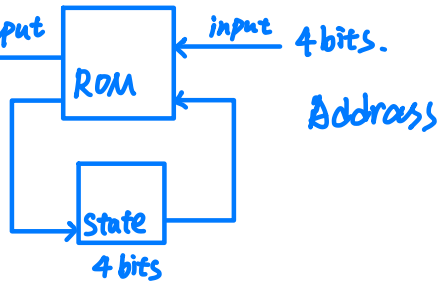
b) Say you have a state machine with 4 bits of input, 3 bits of output, and 12 states How large would your control ROM need to be? Fill in the blanks below. **[6]**

4 bits.

8 *bits of address* 4+4=8

7 *bits of data per address* 3+4=7

A total of $2^{8.7}$ *bits in the ROM.*

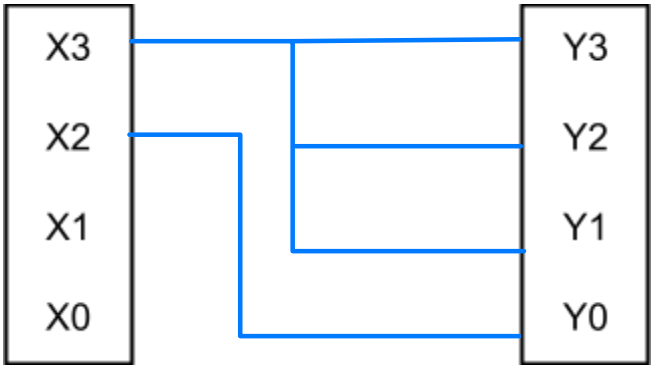


positive → directly right shift 2 bits with 0.
12 ÷4 = 3.
0 1 0 1 0 → 0 0 0 1 0
-16 8 4 2 1
10 ÷4 = 2.5 round → 2.
-10 ÷4 → -3.
1 0 1 1 0
-16 8 4 2 1
1 1 1 0 1
-16 8 4 2 1

Problem 3: Division-by-wire [12 points]

Say you have a 4-bit 2's complement number X[3:0] (so X3 is the most significant bit, X2 is the next most significant, etc.). Using only standard gates (AND, OR, NOT), and constant 0s and 1s, divide X[3:0] by four (round toward negative infinity) and output it as a 4-bit 2's complement number Y[3:0]. Use as few gates as possible. (Hint: you can use *very* few gates indeed.) **(L8)**

right shift 2 bits with 1



Problem 4: I've had my fill of LC2K [8 points]

Using only .fill commands, write an LC2K program which loads the value 5 into reg 1, loads the value 7 into reg 2 and computes bitwise NOR of those two values, leaving the result in register 5 in as few lines as possible (you need to actually have the computer NOR those two values together!) **(L5)**

```
lw 0 1 five
lw 0 2 seven
nor 1 2 5
five .fill 5
seven .fill 7
```

calculate machine code →

```
.fill 8454147
.fill 8519684
.fill 4849669
.fill 5
.fill 7
```

reg1 ← 5

reg2 ← 7

Problem 5: Linking [6 points]

In your own words explain the purpose of using a linker rather than simply compiling the entire program at once. Your answer must be 50 words or fewer and must *briefly* explain how the linker accomplishes that purpose. **(L7)**

Answer:

If we work on a really big program with lots of functionality. If we don't use linker, then once we need to update one functionality. we need to spend lots of time to compile all the program. However, if we have linker, then we only need to compile the modified program and link it back to the big program

Problem 6: Performance [12 points]

(L12)

Two new implementations of the LC2K2 have been proposed.

Implementation 1

- Clock period of 10ns
- Cycles per instruction:

Instruction	Cycles
ADD, NOR, JALR	4 cycles
BEQ	4 cycles
NOOP	2 cycles

Instruction	Cycles
HALT	3 cycles
LW	5 cycles
SW	4 cycles

Implementation 2

- Clock period of 5ns
- Cycles per instruction:

ADD, NOR, LW, SW, JALR	8 cycles
BEQ	10 cycles
NOOP, HALT	1 cycle

Your benchmark consists of 1,000,000 instructions and has the following instruction frequency:

ADD	20%	NOR	20%
BEQ	20%	NOOP	10%
JALR	10%	LW	10%
SW	10%	HALT	0%

a) What is the CPI for implementation 1 on this benchmark? **[3]**

$0.2\times4 + 0.2\times4 + 0.1\times4 + 0.1\times4 + 0.2\times4 + 0.1\times2 + 0.1\times5 + 0\times3$

$= 0.8 + 0.8 + 0.4 + 0.4 + 0.8 + 0.2 + 0.5 + 0 = 3.9$ CPI

b) What is the CPI for implementation 2 on this benchmark? **[3]**

$0.2\times8 + 0.2\times10 + 0.1\times8 + 0.1\times8 + 0.2\times8 + 0.1\times1 + 0.1\times8 + 0\times1$

$= 1.6 + 2 + 0.8 + 0.8 + 1.6 + 0.1 + 0.8 = 7.7$ CPI

c) Which implementation is faster on this benchmark? Briefly explain your answer. **[6]**

implementation 1: $3.9 \times 10ns \times 1\times10^6 = 3.9\times10^7$ ns

implementation 2: $7.7 \times 5ns \times 1\times10^6 = 3.85\times10^7$ ns

$3.85\times10^7 ns < 3.9\times10^7 ns$.

So implementation 2 is faster on this benchmark.

Problem 7: Pushing multicycle [24 points]

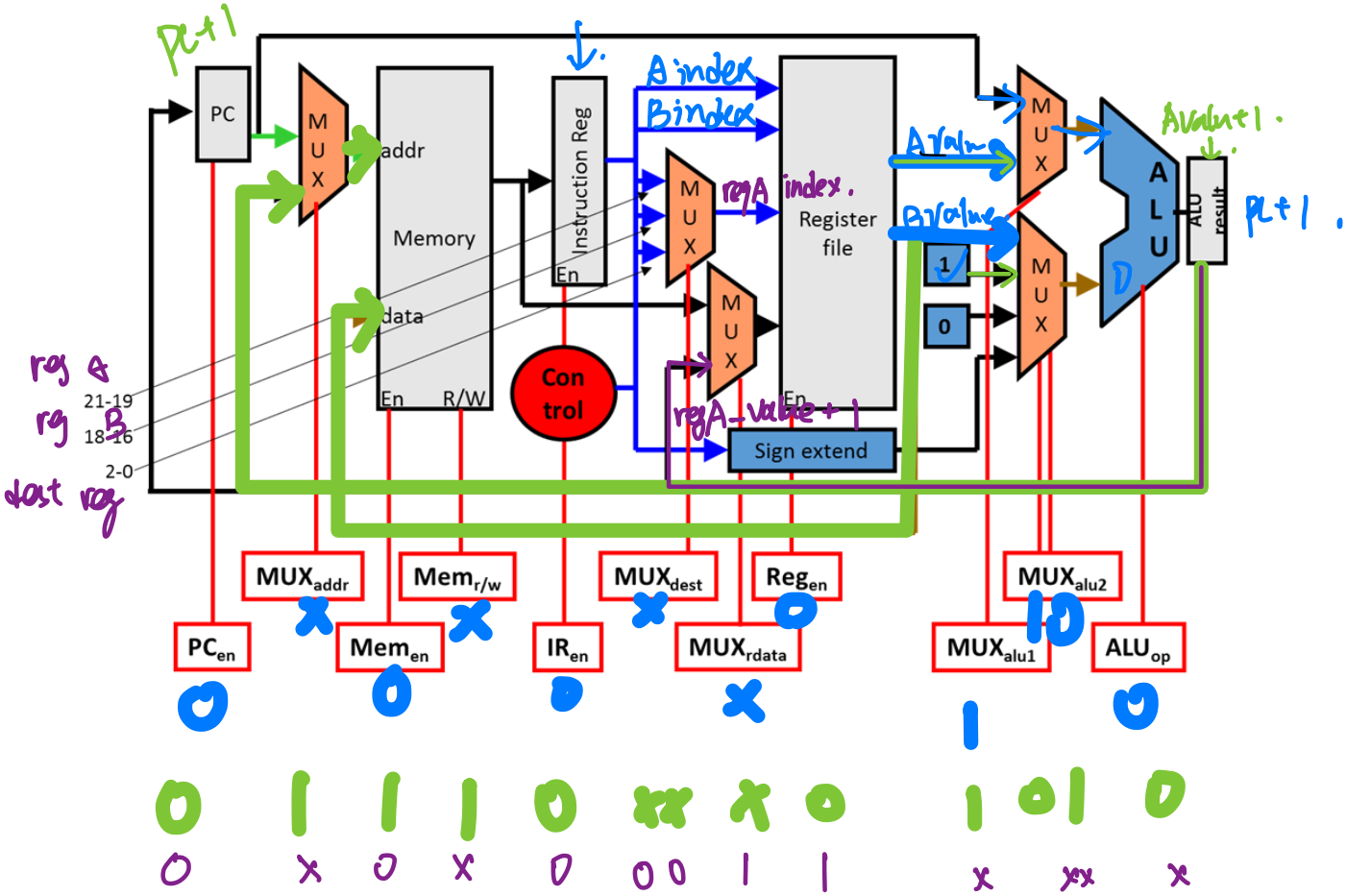
In a moment of clarity, the EECS 370 staff realizes that the LC2K instruction set is rather limited when compared to its greatest marketplace rival, ARM. In order to add convenience and attract users, the EECS 370 staff decides to implement two new instructions: push and pop. **(L11)**

`push regA regB`
 	
`//MEM[REG[regA]] = REG[regB];`
 	
`// REG[regA]++;`
	`pop regA regB`
	
	`// REG[regA]--;`
	
	`// REG[regB] = MEM[REG[regA]];`

Using the modified multi-cycle LC2K datapath below, fill out the control ROM table for the **push instruction**, using as few cycles as possible. Write "X" if the setting doesn't matter. Multiplexers' inputs are numbered 0, 1, 2… where 0 is the topmost input. You are to use as few cycles as possible. You will not be modifying the datapath given. (*Note: for this problem you are not doing anything with the pop instruction*).

Notes:

- regA is intended to be the stack pointer
- regB is the data to be pushed or popped the stack
- The first two cycles, FETCH and DECODE, have been given to you
- Several entries in the table require multiple control bits
- There may be unused rows in the table
- Recall:
  0 = Read and 1 = Write for Mem r/w
  0 = ADD and 1 = NOR for ALUop



cycle 3: calculate A value and store it in ALU register.

cycle 4: path the index into memory and chang the value at that location
Also calculate reg[regA]++.

cycle 5: Store the new value in regA.

Cycle	PC (en)	MUX (addr)	Mem (en)	Mem (r/w)	IR (en)	MUX (dest)	MUX (rdata)	Reg (en)	MUX (alu1)	MUX (alu2)	ALU (op)
1	0	0	1	0	1	X	X	0	0	01	0
2	1	X	0	X	0	X	X	0	X	X	X
3	0	x	0	x	0	xx	x	0	1	10	0
4	0	1	1	1	0	xx	x	0	1	01	0
5	0	x	0	x	0	00	1	1	x	xx	x
6											
7											
8											

Problem 8: SC/MC Datapath Performance [22 points, Group]

Consider the multi-cycle and single-cycle datapath from lecture but with the delays shown for each operation. **(L11)**

Read memory	10 ns
Write memory	15 ns
Read register file	3 ns
Write register file	6 ns
ALU	6 ns
All other operations	0 ns

Add : Read from instruction memory + Read from register file + ALU(add) + write into register file.
10ns + 3ns + 6ns + 6ns = 25ns

nor : Read from instruction memory + Read from register file + ALU(nor) + write into register file.
10ns + 3ns + 6ns + 6ns = 25ns

beq : Read from instruction memory + Read from register file + ALU(comp)
10ns + 3ns + 6ns = 19ns

sw : Read from instruction memory + Read from register file + ALU(add) + write into memory.
10ns + 3ns + 6ns + 15ns = 34ns

(a) What is the clock period for the single-cycle processor if we only had to support add, nor, beq and sw? **[4]**

So the clock period for the single-cycle processor is 34ns

(b) What is the clock period for the single-cycle processor if we support all LC2K instructions except jalr? **[4]**

lw : Read from instruction memory + Read from register file + ALU(add) + read from data memory + write into Register file.
10ns + 3ns + 6ns + 10ns + 6ns = 35ns,

35ns, since the clock period depends on the longest instruction.

(c) If we can decrease the delay for one of the operations by 10%, which operation should it be for our **single-cycle** datapath which supports all LC2K instructions except jalr? What is the clock period after improvement? **[7]**

"Read from memory": 10ns → After Change → 9ns.

sw will change to 33ns
lw will change to 33ns

clock period after improvement should be 33ns.

(d) If we can decrease the delay for one of the operations by 10%, which operation should it be for our **multi-cycle** datapath which supports all LC2K instructions except jalr? What is the clock period after improvement? **[7]**

We only need to consider the longest subcycle which is write memory here: 15,

15 × 0.9 = 13.5ns.

So the new clock should be 13.5ns.